Microsoft
.netdeveloper
training

The .NET Languages

Agenda: Day 1

Module 01: What is Microsoft® .NET?

Module 02: Visual Studio® .NET

▶ Module 03: The .NET Languages

Module 04: Building ASP®.NET Applications

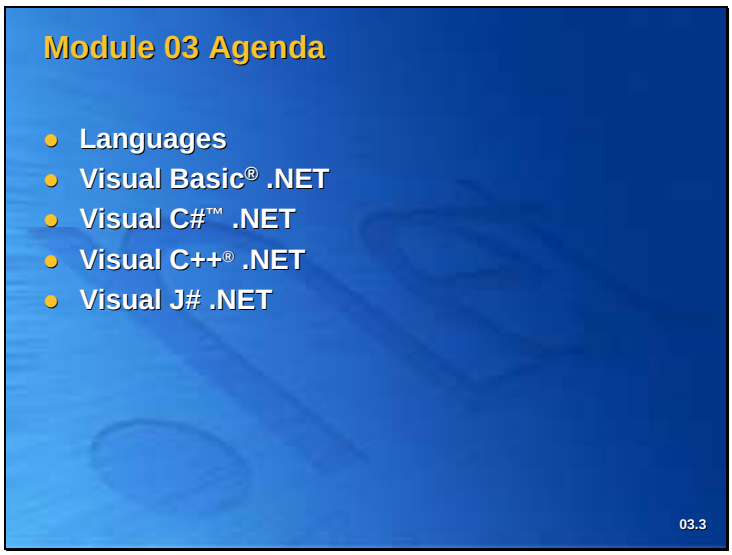
Module 05: Using Data Access in .NET Applications

Module 06: Object Oriented Concepts in .NET

Module 07: Creating Windows Applications

Module 08: Using XML in .NET

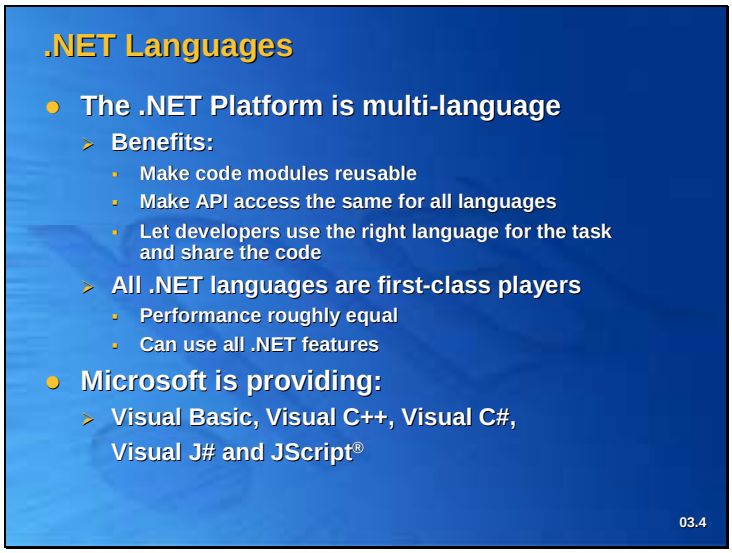
 ©2002 Microsoft Corporation



Module 03 Agenda

What you will learn:

- Everything is an object in .NET and all languages can use .NET features
- You have lots of choices about what language to use and that enables you to use the language that works best for the job at hand


.NET Languages
The .NET Platform is multi-language
Benefits:
Make code modules reusable
Make API access the same for all languages
Let developers use the right language for the task and share the code
All .NET languages are first-class players
Performance roughly equal
Can use all .NET features
Microsoft is providing:
Visual Basic, Visual C++, Visual C#, Visual J# and JScript®
03.4

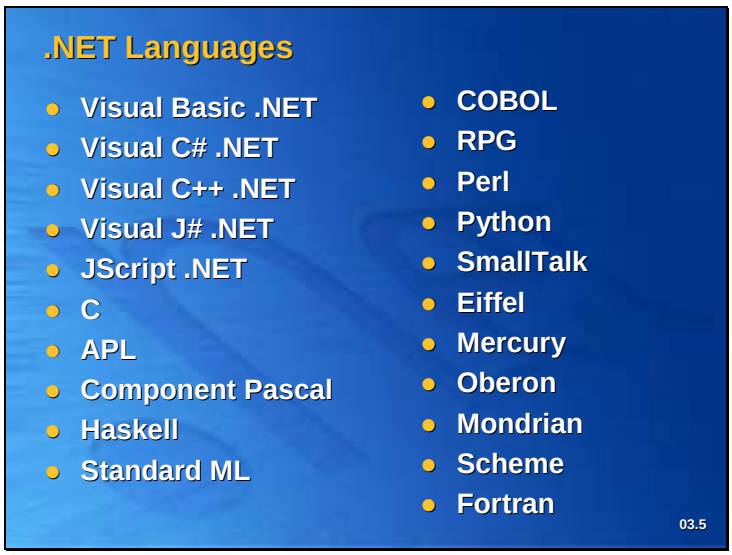
.NET and Languages

In the past, developers had problems sharing code since some worked in Visual C++ and others in Visual Basic. Depending upon how the Visual C++ code is written, it may be reusable for the Visual Basic developers. Microsoft created .NET to alleviate this problem and today, any developer can use a .NET class and event by extending that class for their own purposes. .NET languages build on the common type system of the .NET Framework. Visual Basic .NET and Visual C# provide native access to the .NET type system.

Core concepts:

- Everything is an object
- Everything implicitly inherits from System.Object
- There is a clear distinction between value and reference types
    - By-Value: Simple Types, Enumerations and Structs
    - By-Reference: Interfaces, Classes and Arrays
- All languages can use .NET features

 ©2002 Microsoft Corporation

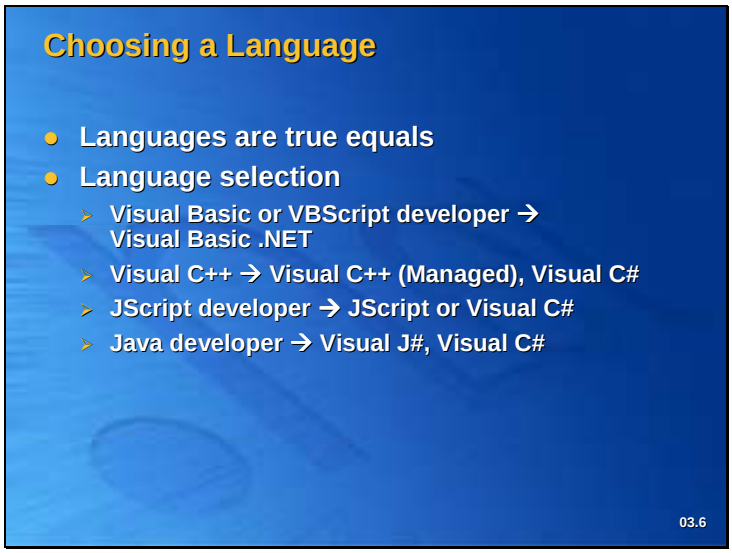


.NET Languages

The .NET platform is a multi-language platform and all the languages share common features.

Everything is an object:

- Unified Type System insures simplicity at the lowest levels
- All languages use the Unified Class Libraries
- All languages have: Properties, Methods, Events, Attributes and XML documentation
- Can be embedded in ASP.NET pages


Choosing a Language
Languages are true equals
Language selection
Visual Basic or VBScript developer → Visual Basic .NET
Visual C++ → Visual C++ (Managed), Visual C#
JScript developer → JScript or Visual C#
Java developer → Visual J#, Visual C#
03.6

Choosing a Language

To guarantee the usability of your components and controls from other programming languages, you must author them in a Common Language Specification (CLS) compliant language and ensure that all public and protected members are CLS compliant.

The **.NET Framework SDK** provides compilers for five CLS-compliant languages — Visual Basic .NET, Visual J#, C#, Managed Extensions for Visual C++ and JScript.

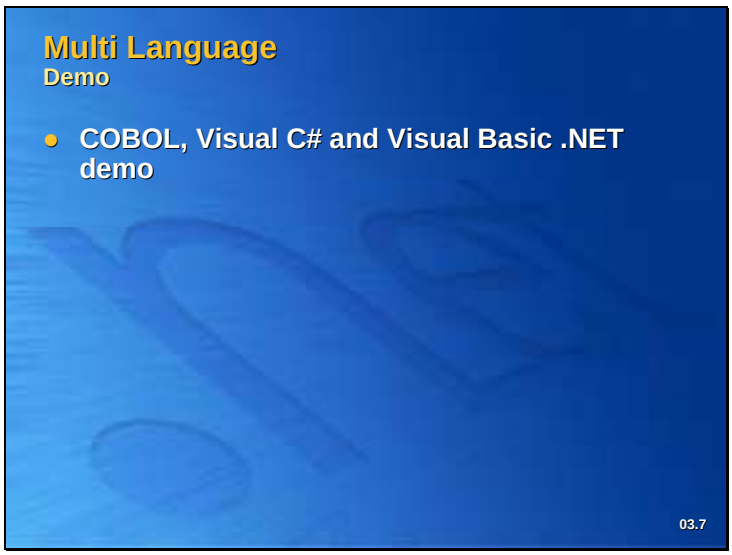
All .NET languages use the same runtime, and that gives them all the same performance and same application feature set. But the languages themselves were designed for different target developers.

The Visual Basic .NET language has classic-Visual Basic features like declarative events, dynamic arrays, case insensitivity, better COM interoperability, late binding, modules, optional parameters, and a background compiler for immediate error updates in the editor. These make the Visual Basic language much better suited to typical RAD developers.

Conversely, C# supports syntax for unsafe pointers, operator overloading, case sensitivity, and bitwise shift operations since their target users include lower-level C developers.

For the Java developer, .NET provides Visual J#. Visual J# uses the language syntax, and is fully integrated with Visual Studio .NET development environment and the .NET Framework, providing support for .NET features such as XML Web services, ASP.NET, ADO.NET, and Windows Forms.

 ©2002 Microsoft Corporation

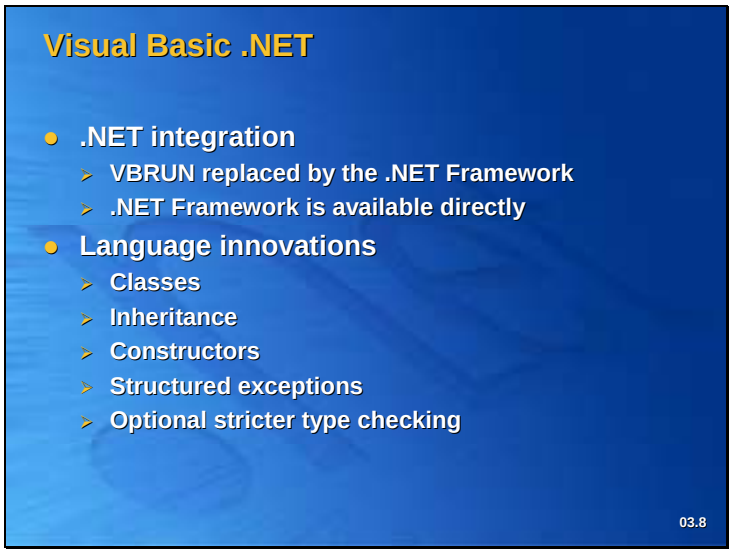


Languages

Demo

The purpose of this demo is to illustrate:

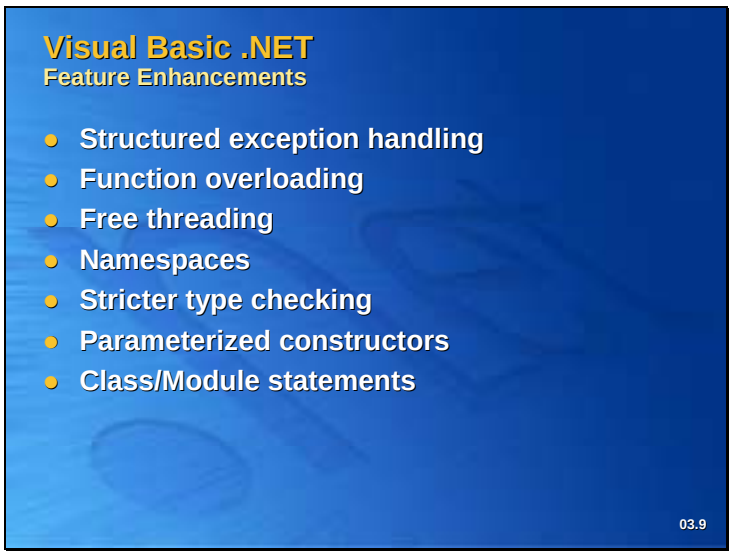
- COBOL, C#, Visual Basic .NET demo



Visual Basic .NET

Visual Basic .NET provides developers with access to a richer and more powerful toolset than in previous versions of Visual Basic. Driven by customer demand, Visual Basic .NET delivers an extensive set of features including full object-oriented design capabilities, free-threading and direct access to the .NET Framework.

 ©2002 Microsoft Corporation



Visual Basic .NET

Feature Enhancements

Structured exception handling - Visual Basic .NET like all other .NET languages offers centralized error handling and it supports a **try…catch…finally** construct similar to C++. This feature allows you to put a sub-routine at the top of your code and let the errors bubble into it.

Function overloading - Overloading is the process of creating two or more functions with the same name, but with different function signatures. With overloading, you can implement functions that react differently based on input type.

Free threading - Visual Basic 6.0 supports an apartment-threaded model that provides some efficiency to deployed applications, but stops short of ideal. Visual Basic .NET offers a free-threaded model that makes a difference when writing scalable Web Applications. In addition, Visual Basic .NET includes language constructs that let you spawn multiple threads.

More information on future enhancements is available at:

http://msdn.microsoft.com/msdnmag/issues/0400/vbnexgen/vbnexgen.asp

Namespaces - A logical naming scheme for grouping related types. For example, the .NET Framework uses a hierarchical naming scheme for grouping types into logical categories of related functionality, such as ASP.NET application framework, or remoting functionality.

Stricter type checking – Traditionally, Visual Basic has had permissive semantics; that is, the Visual Basic language in general does not require explicit syntax to be used when performing operations that might not be optimally efficient (for example, late binding) or that might fail at runtime (for example, narrowing conversions). Although permissive semantics simplifies learning the Visual Basic language, it can hamper large-scale application development by allowing coding errors to slip by undetected. Visual Basic .NET provides the option of enforcing strict semantics during compilation.

Parameterized constructors – Visual Basic .NET revises object creation for improved readability and supports constructors with arguments.

In Visual Basic 6.0, an object variable declared with the keywords As New is initialized to Nothing, meaning that no object has been created. Every time the variable is encountered during execution, it is evaluated before the code executes. If the variable contains Nothing, an object of the appropriate class is created before the code that uses it executes.

In Visual Basic .NET, there is no implicit object creation. If an Object Variable contains Nothing when it is encountered, it remains unchanged and no instance is created.

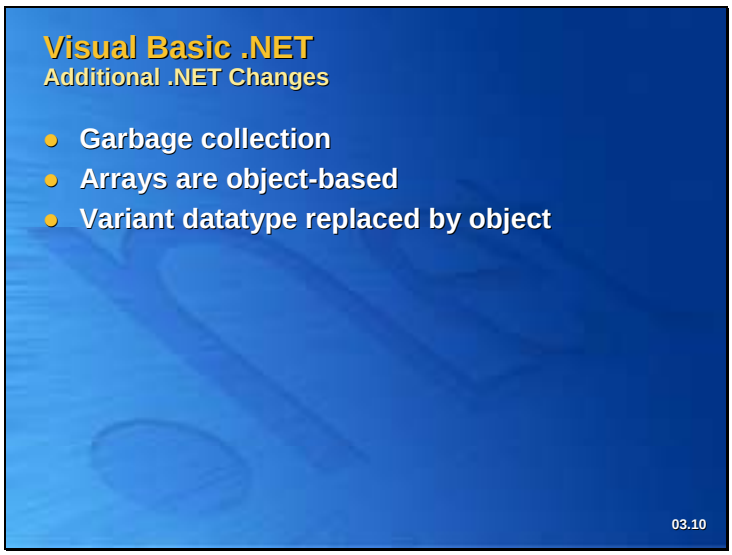
You can create an object with the same statement that declares the object variable. The following two lines of code are treated as equivalent, with the first being taken as shorthand for the second:

```
Dim Emp As New EmpObj() ' Shorthand for next line
Dim Emp As EmpObj = New EmpObj() ' EmpObj created; Emp points to EmpObj
```

In both cases, an EmpObj object is created as soon as the Dim statement is executed, and Emp is initialized to a reference to the new object.

Class/Module statements - Instead of using separate files for classes and modules as in Visual Basic 6.0, Visual Basic .NET allows you to have multiple classes or modules in the same file. A class is defined by the appropriate Class statement while the Module statement defines a block of code.

 ©2002 Microsoft Corporation



Visual Basic .NET

Additional .NET Changes

Garbage collection – Periodically your program will pause to conduct garbage collection. At that time, the system looks through the global variables and stack to determine which Objects are currently being referenced. It also walks references in the Objects themselves to properly account for linked lists and other linked data structures. Being able to know where the references are in your Objects is one of the reasons .NET Framework data is strongly typed.

The unused Objects are freed (if they do not need finalization) or put in a list for finalization. This extra step for finalization is why you should avoid writing finalizers you do not need. After the Objects are finalized, they are freed in some future garbage collection.

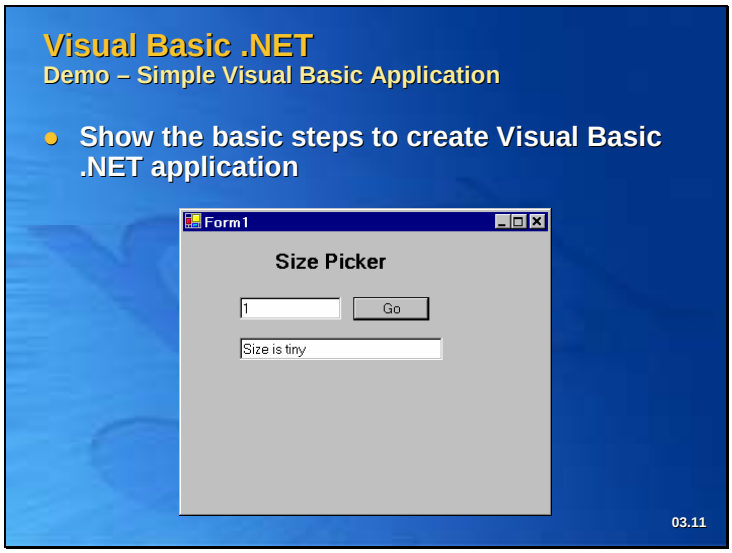
Arrays are Object-based - In .NET, everything is an Object.

Variant datatype replaced by object:

- **Variant to Object Conversion -** Previous versions of Visual Basic supported the Variant datatype, which is no longer supported in Visual Basic .NET. All variables declared as Variant are converted to variables of type Object.

Also new to Visual Basic .NET is:

- **Integer to Short, Long to Integer -** In Visual Basic 6.0, the datatype for 32-bit whole numbers was Long; the datatype for 16-bit whole numbers was Integer. In Visual Basic .NET, Long is now the datatype for 64-bit whole numbers; Integer is the datatype for 32-bit whole numbers; and Short is the datatype for 16-bit whole numbers. The Upgrade Tool automatically converts 32-bit and 16-bit datatypes to Integer and Short, respectively.
- **Parameterless Default Properties -** In Visual Basic 6.0, many Objects exposed default properties, which can be omitted as a programming shortcut. The Upgrade Tool will resolve these default properties that do not have parameters, for use in Visual Basic .NET.
- **Non-Zero Bound Arrays -** Visual Basic 6.0 allowed the use of arrays with lower and upper bounds of any whole number. The Visual Basic .NET Upgrade Tool upgrades non-zero bound arrays using an array wrapper class.



Visual Basic .NET

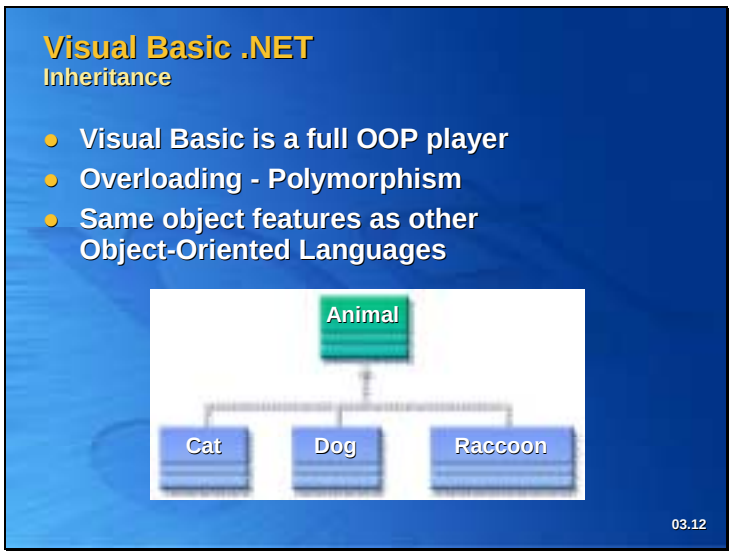
Demo

The purpose of this demo is to show the creation and functionality of a simple Visual Basic Application.

Simple Visual Basic Application

- Create a simple Visual Basic application
- Show the basic steps to create Visual Basic .NET application
- Walk through simple functionality
- Explain similarities to Visual Basic 6 and new features such as the common Message Box

 ©2002 Microsoft Corporation



Visual Basic .NET

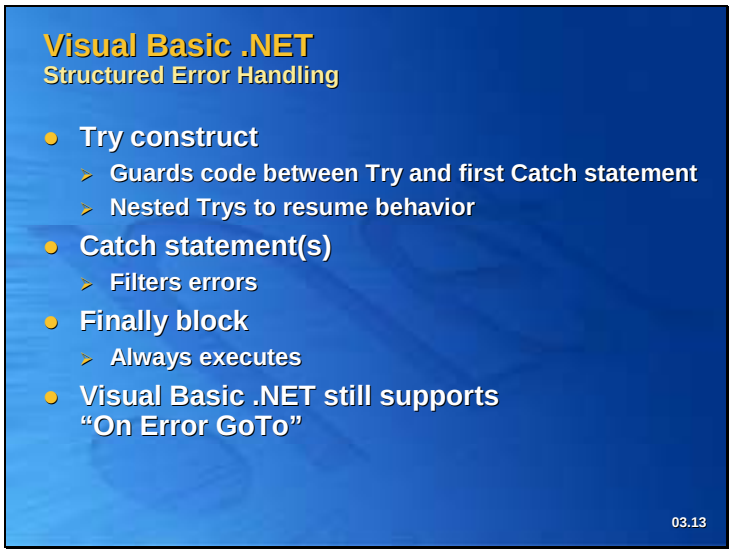
Inheritance

Visual Basic .NET is the first Visual Basic programming language that is completely Object-oriented.

Because developing for the Internet requires rapid assembly and massive reuse, support for implementation inheritance was high on the request list for Visual Basic. Visual Basic .NET now supports visual form inheritance and other implementation inheritance features.

Developers can use the keyword **Inherits** to derive from an existing class. Instances of the derived class support and can extend all methods and interfaces supported by the base class.

In addition, the Visual Basic language has been streamlined by the exclusion of legacy keywords, and the inclusion of features such as type safety, and low-level constructs required by advanced developers. Visual Basic .NET also supports **Inheritance, polymorphism and overloading**, along with other new constructs including structured error handling and free threading.



Visual Basic .NET

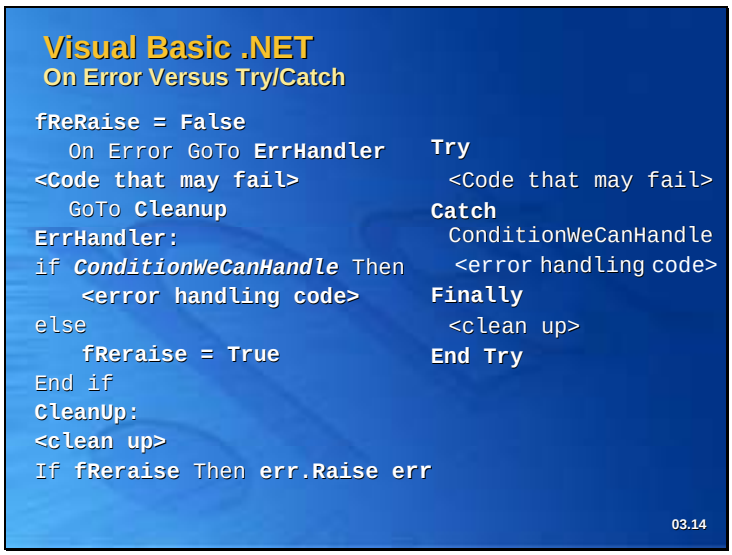
Structured Error Handling

Developing Enterprise Applications requires the construction of reusable, maintainable components. One challenging aspect of the Basic language in past versions of Visual Basic was its support for error handling, which was found to involve duplicated code. Error handling using the existing **On Error GoTo** statement sometimes slows the development and maintenance of large applications and its name reflects some of these problems. As the **GoTo** implies, when an error occurs control is transferred to a labeled location inside the sub-routine. Once the error code runs, it often must be diverted to another cleanup location by using a standard **GoTo**, which uses yet another **GoTo** or an **Exit** out of the procedure. Handling different errors having a variety of combinations of **Resume** and **Next** quickly produces illegible code and it leads to frequent bugs when execution paths are not thought out completely.

With **Try ... Catch ... Finally**, these problems are resolved so developers can nest their exception handling and there is a control structure for writing clean-up code that executes in both normal and exception conditions.

- Eliminates common need for GoTo.
- Throw statements
- Supports typed exceptions

 ©2002 Microsoft Corporation

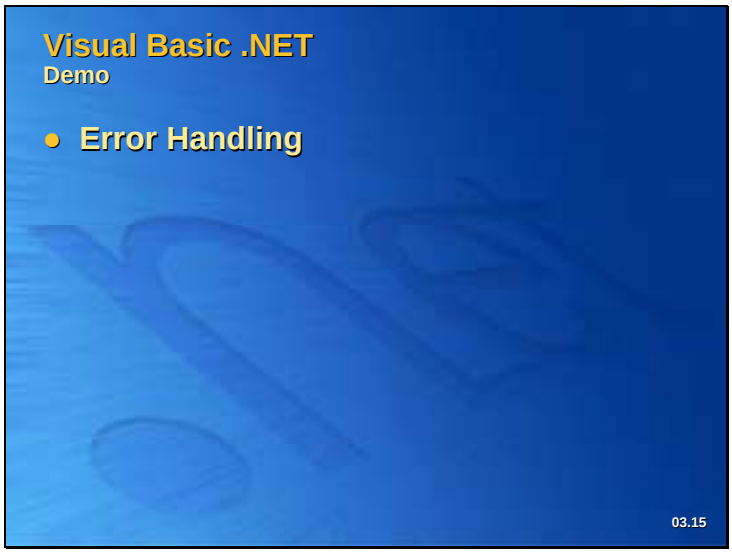


Visual Basic .NET

On Error Versus Try/Catch

While **Error handling** in Visual Basic proved to be insufficient, the new constructs in Visual Basic .NET allow the developer to build robust error handling with relative ease.

- Explicit code
  - Code is more elegant, readable and efficient
- Scenario – Opening/processing file … when finished or in case of an error, close file
- Step through code
  - Any number of catch statements
  - <clean up> code is what closes the file



Visual Basic .NET

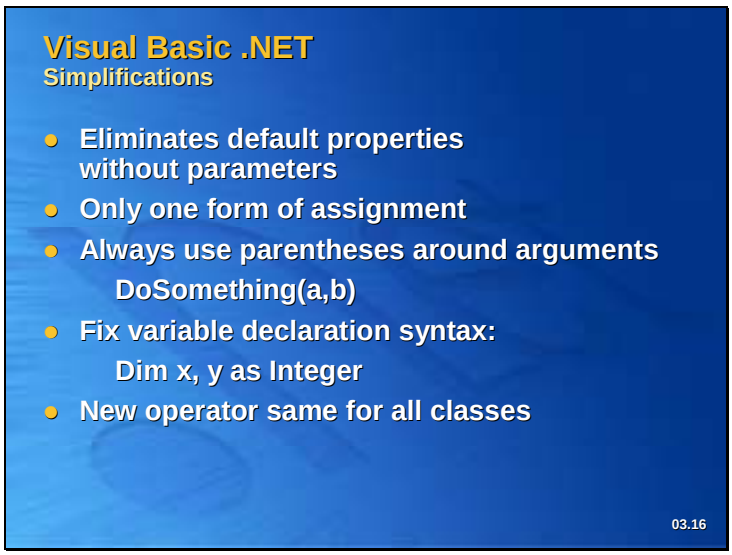
Demo

The purpose of this demo is to illustrate:

Error Handling

- Basic error handling
- How to raise errors
- Using Try/Catch/Finally/End Try , vs. On Error GoTo

 ©2002 Microsoft Corporation

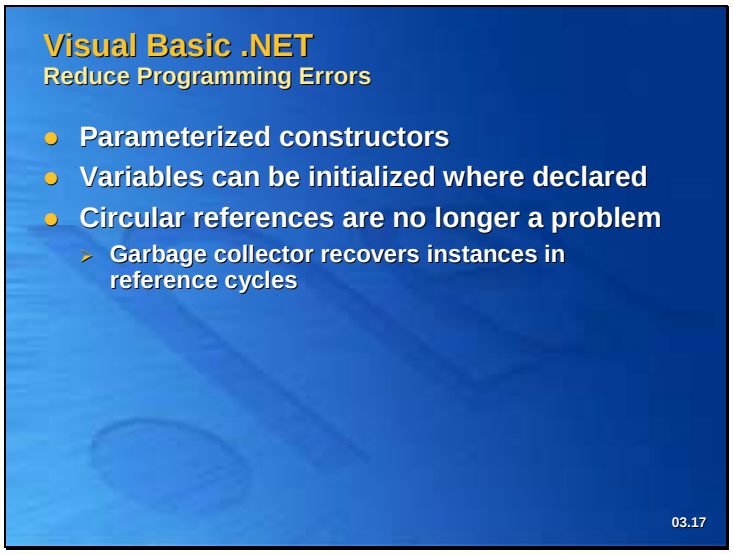


Visual Basic .NET

Simplifications

This slide offers an example of using short cuts in expressions.

Note: It is possible to put multiple type declarations in one Dim but it is recommended that you do not.


Visual Basic .NET
Reduce Programming Errors
Parameterized constructors
Variables can be initialized where declared
Circular references are no longer a problem
Garbage collector recovers instances in reference cycles
03.17

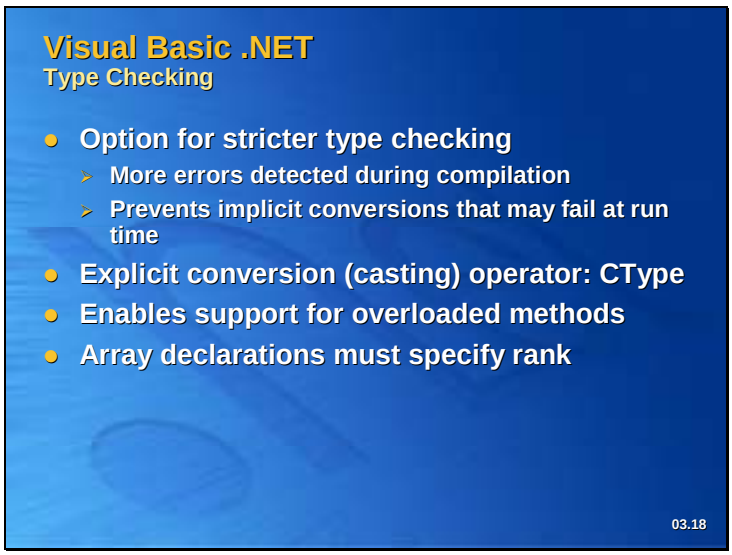
Visual Basic .NET

Reduce Programming Errors

See slide 9 for details on Parameterized constructors, Variables initialization and declarations.

See slide 10 for details on Garbage collection.

 ©2002 Microsoft Corporation


Visual Basic .NET
Type Checking
Option for stricter type checking
More errors detected during compilation
Prevents implicit conversions that may fail at run time
Explicit conversion (casting) operator: CType
Enables support for overloaded methods
Array declarations must specify rank
03.18

Visual Basic .NET

Type Checking

Today, the Visual Basic language is liberal in the implicit type coercions it generates. For assignment and for parameter passing other than by reference, the Visual Basic compiler allows nearly any datatype to be converted to any other by generating run time coercion, but the run time coercion operation fails if the value to be converted cannot be without data loss. Through the addition of a new compilation option, Visual Basic can generate compile-time errors for any conversions that may cause an error at run time. **Option strict** improves type safety by generating errors when a conversion is required which could fail at run time or which, like the automatic conversion between numeric types and strings, is unexpected.

Stricter type checking – Traditionally, Visual Basic has had permissive semantics; that is, the Visual Basic language in general does not require explicit syntax to be used when performing operations that might not be optimally efficient (for example, late binding) or that might fail at run time (for example, narrowing conversions). Although permissive semantics simplifies learning the Visual Basic language, it can hamper large-scale application development by allowing coding errors to slip by undetected. Visual Basic .NET offers the option of enforcing strict semantics during compilation.

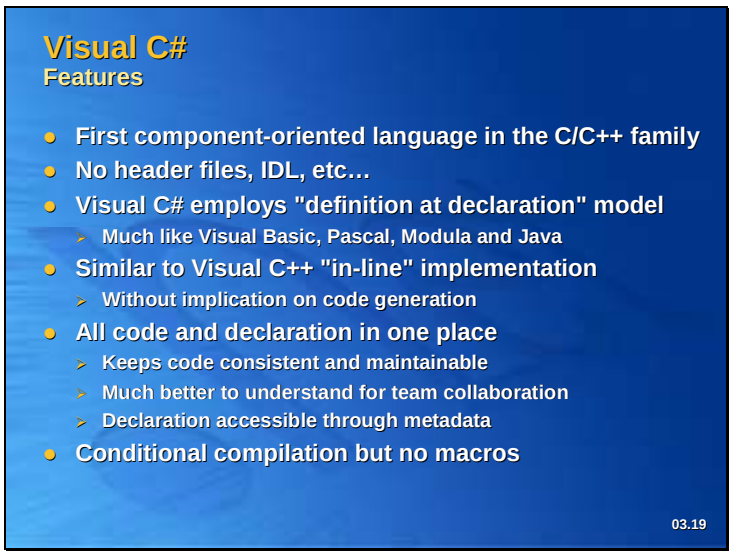
The **Option Strict** statement determines whether conversions and operations on **Object** are governed by strict or permissive type semantics and whether types are implicitly **Object** by default. The statement may be followed by the keywords **On** or **Off**; the default is **Off**. If no statement is specified in a file, the compilation environment determines which will be used.

Under strict semantics, the following are disallowed:

- Narrowing conversions without an explicit cast operator.
- Late binding.
- Operations on type **Object** other than **=**, **<>**, **TypeOf...Is**, and **Is**.
- Omitting the **As** clause in a declaration.

Note: **Strict** and **Off** are not reserved words. *OptionStrictDirective* ::= **Option Strict** [*OnOff*] *LineTerminator*

 ©2002 Microsoft Corporation


Visual C#
Features
First component-oriented language in the C/C++ family
No header files, IDL, etc…
Visual C# employs "definition at declaration" model
Much like Visual Basic, Pascal, Modula and Java
Similar to Visual C++ "in-line" implementation
Without implication on code generation
All code and declaration in one place
Keeps code consistent and maintainable
Much better to understand for team collaboration
Declaration accessible through metadata
Conditional compilation but no macros
03.19

Visual C#

Features

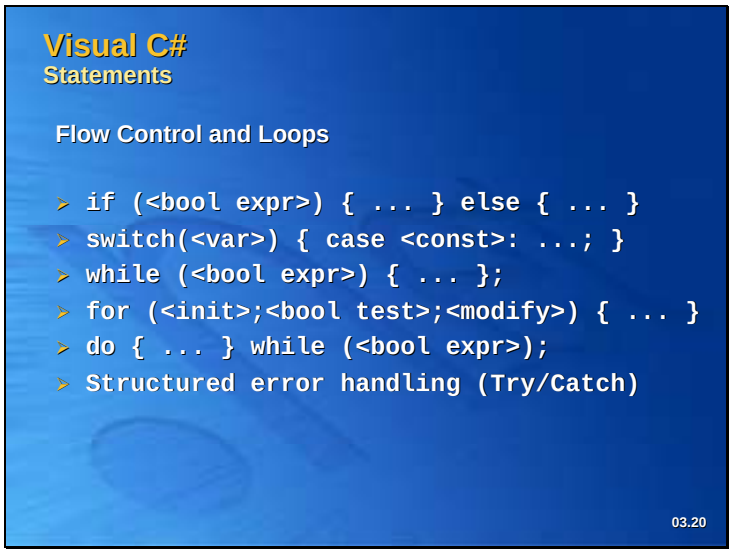
Microsoft built Visual C# with type-safety, garbage collection, simplified type declarations, versioning and scalability support, and a number of other features that make developing solutions faster and easier, especially for Enterprise Services and Web services.

Consistency

Visual C# unifies the type system by letting you view every type in the language as an Object. Whether you are using a class, a struct, an array or a primitive, you will be able to treat it as an Object. Objects are combined into namespaces, which allow you to access everything programmatically.

The design of Visual C# eliminates the most common C++ programming errors. For example:

- Garbage collection relieves the programmer of the burden of manual memory management.
- Variables in C# are automatically initialized by the environment.
- Variables are type-safe.



Visual C#

Statements

Supports structured error handling

Structure

Try running this code ...

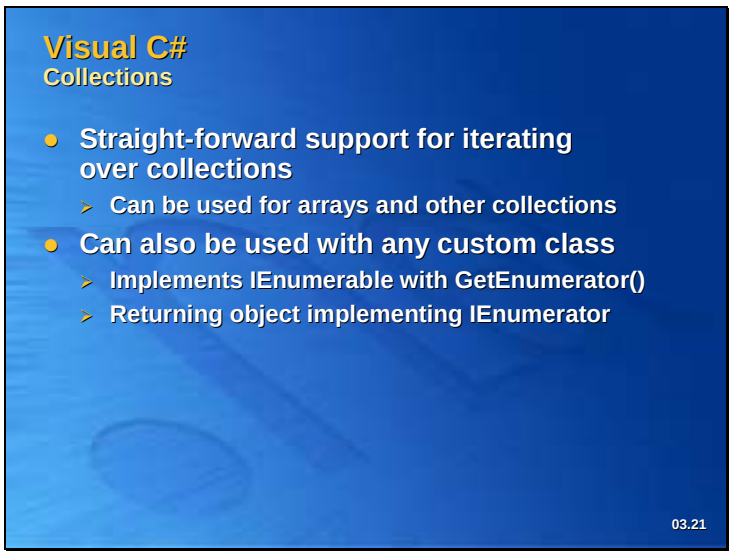
... if error occurs, *Catch* what I can deal with ...

...*Finally* allow me to do some manual cleanup work

Example:

```
try
{
   //... run code
}
catch(SomeException e)
{
   //... handle
}
finally
{
   //...end gracefully
}
```

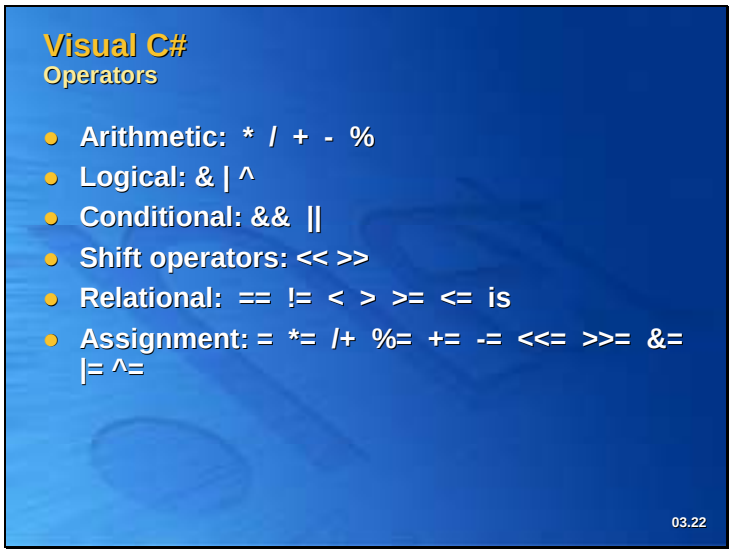
 ©2002 Microsoft Corporation


Visual C#
Collections
Straight-forward support for iterating over collections
Can be used for arrays and other collections
Can also be used with any custom class
Implements IEnumerable with GetEnumerator()
Returning object implementing IEnumerator
03.21

Visual C#

Collections

```
int odd = 0, even = 0;
int[] arr = new int [] {0,1,2,5,7,8,11};
foreach (int i in arr)
{
  if (i%2 == 0)
      even++;
  else
      odd++;
}
Console.WriteLine("Found {0} Odd Numbers, and {1} Even Numbers.", odd, even);
```

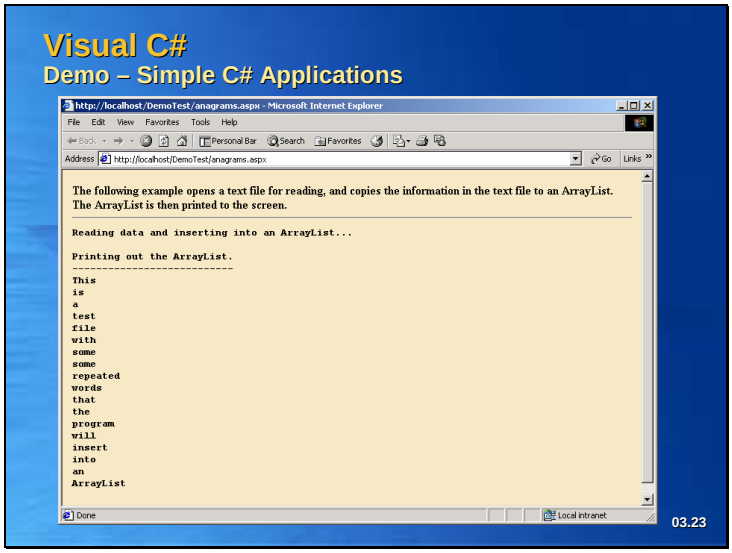


Visual C#

Operators

C# provides a large set of operators, which are symbols that specify which operations to perform in an expression. C# predefines the usual arithmetic and logical operators, as well as a variety of others as shown in the slide. In addition, many operators can be overloaded by the user, thus changing their meaning when applied to a user-defined type.

 ©2002 Microsoft Corporation



Visual C#

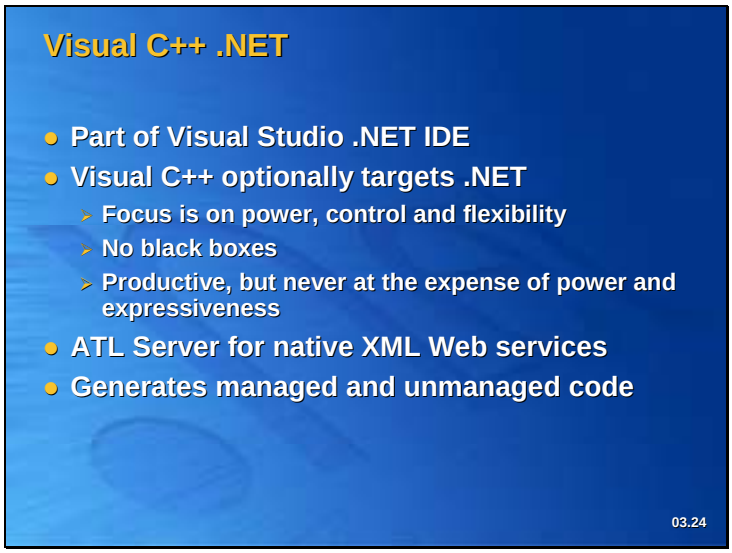
Demo

The purpose of this demo is to illustrate the creation and functionality of a simple Visual C# application.

Simple C# Application

Create simple C# application to demonstrate:

- How C# is similar to C++ and JScript languages?
- Discuss how C# has access to entire .NET Framework


Visual C++ .NET
Part of Visual Studio .NET IDE
Visual C++ optionally targets .NET
Focus is on power, control and flexibility
No black boxes
Productive, but never at the expense of power and expressiveness
ATL Server for native XML Web services
Generates managed and unmanaged code
03.24

Visual C++ .NET

Visual C++ .NET is a fully incorporated member of Visual Studio .NET. It shares many of the same editors, tools and paradigms; however, it does not share the Windows Forms editor.

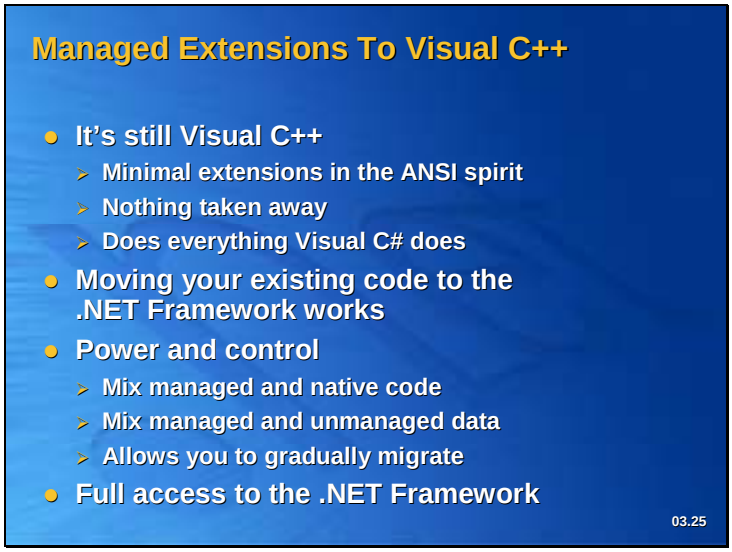
Visual C++ .NET optionally targets .NET managed assemblies. That is to say, you can use Visual C++ to continue to target native Windows Applications. It is the only Microsoft language/environment that natively targets Windows.

As always, Visual C++ continues to focus on power, control and flexibility. It ships full source code to the included native libraries and gives you low-level access to everything. This makes the language more complicated, but sometimes the power is necessary.

ATL Server is a set of new classes in the ATL Library that enable you to create native XML Web services that are compatible with managed XML Web services. You may choose the ATL Server if you have an existing ISAPI application you want to enable with XML Web services.

Visual C++ now lets you target managed code as well as Managed Extensions.

 ©2002 Microsoft Corporation


Managed Extensions To Visual C++
It's still Visual C++
Minimal extensions in the ANSI spirit
Nothing taken away
Does everything Visual C# does
Moving your existing code to the .NET Framework works
Power and control
Mix managed and native code
Mix managed and unmanaged data
Allows you to gradually migrate
Full access to the .NET Framework
03.25

Managed Extensions to Visual C++

Managed Extensions for Visual C++ now include keywords that conform to ANSI guidelines (named with double underbars __nnn). These keywords are similar to many of the native keywords in Visual C#. With these keywords, you can do most of what you can do in Visual C#, though some of it will be at a lower level. With the extension keywords, you can incorporate managed data in the application.

Visual C++ enables you to seamlessly mix both native and managed code and data in a single image. The compiler automatically generates code to context-switch between the two environments.

With managed extensions the C++ developer can leverage and incorporate the full .NET Framework provided base-class services.

Extensive interoperability - The managed, type-safe environment is appropriate for most Enterprise Applications, but real-world experience shows that some applications continue to require native code either for performance reasons or to interoperate with existing Application Programming Interfaces (APIs). Such scenarios may require some developers to use Visual C++.

Visual C# addresses these problems by:

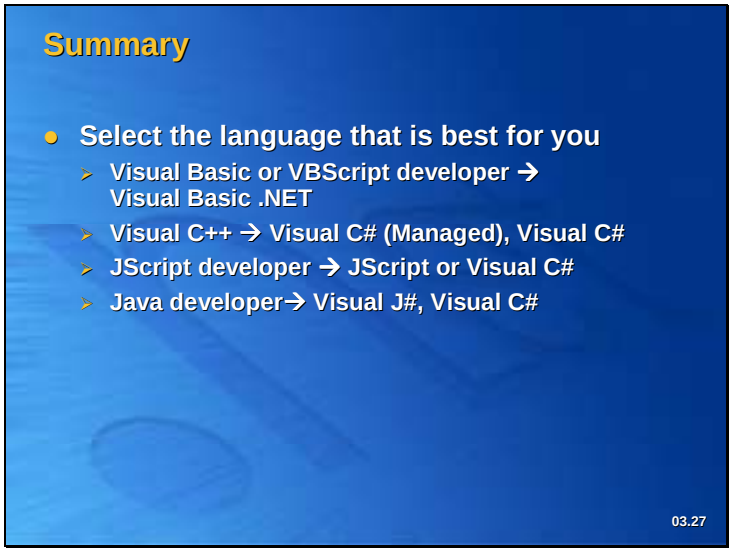
- Including native support for the Component Object Model (COM) and Windows®-based APIs
- Allowing restricted use of native pointers

Visual J#

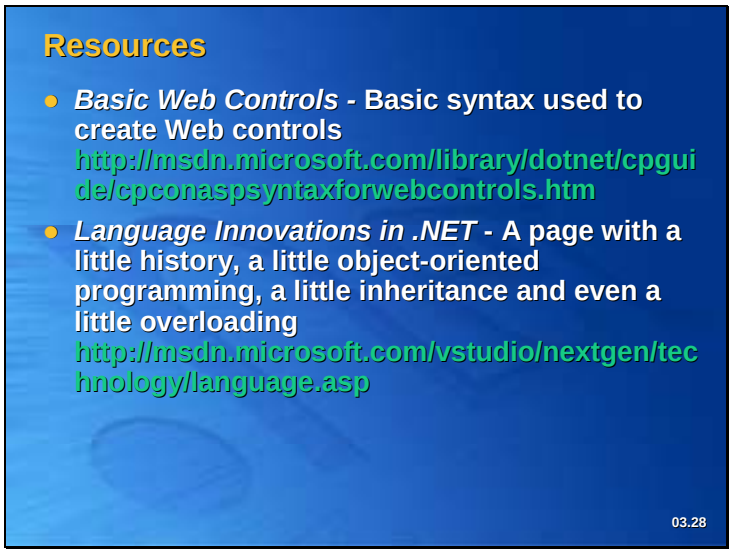
Visual J# is Microsoft's Java language tool for the .NET Frame work. You can use Visual J# to create applications that exploit the .NET Framework libraries. You can also use the class libraries to create applications. Microsoft Visual J# provides the same level of JDK 1.1.4 class library support as Visual J++ 6.0 provides. In addition Visual J# also supports Microsoft extensions such as Windows Foundation Classes (WFC), J/Direct and /COM integration that Visual J++ 6.0 supports.

Existing application sources can be converted to MSIL by using the language source to MSIL Converter. These applications can make calls into class libraries such JDBC as well as Microsoft extensions such as WFC. Applications available only in bytecode format can also be converted to MSIL too, by using the bytecode to MSIL converter.

 ©2002 Microsoft Corporation


Summary
Select the language that is best for you
Visual Basic or VBScript developer → Visual Basic .NET
Visual C++ → Visual C# (Managed), Visual C#
JScript developer → JScript or Visual C#
Java developer→ Visual J#, Visual C#
03.27

Summary


Resources
Basic Web Controls - Basic syntax used to create Web controls
http://msdn.microsoft.com/library/dotnet/cpguide/cpconaspsyntaxforwebcontrols.htm
Language Innovations in .NET - A page with a little history, a little object-oriented programming, a little inheritance and even a little overloading
http://msdn.microsoft.com/vstudio/nextgen/technology/language.asp
03.28

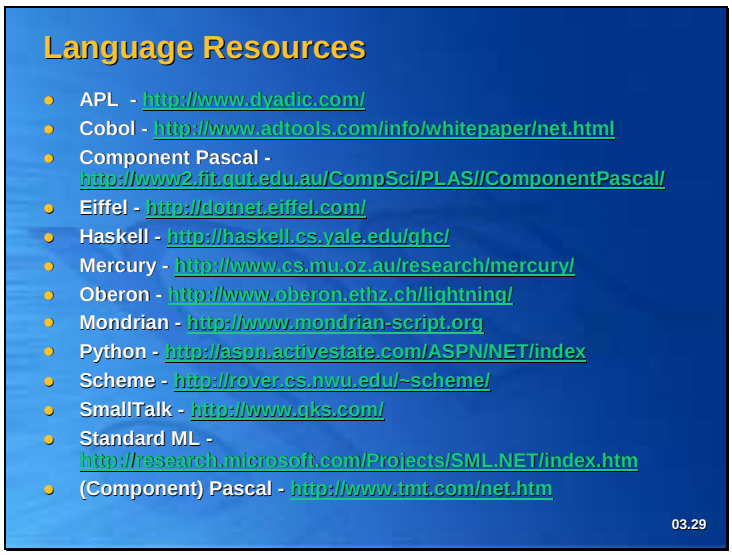
Resources

A number of great features, such as J/Direct and Java-COM integration, have allowed developers to use the Java language while taking full advantage of the Windows platform.

Basic Web Controls - Basic syntax used to create Web controls
http://msdn.microsoft.com/library/dotnet/cpguide/cpconaspsyntaxforwebcontrols.htm

Language Innovations in .NET - A page with a little history, a little object-oriented programming, a little inheritance and even a little overloading
http://msdn.microsoft.com/vstudio/nextgen/technology/language.asp

 ©2002 Microsoft Corporation



Language Resources

APL - http://www.dyadic.com/

COBOL - http://www.adtools.com/info/whitepaper/net.html

Component Pascal - http://www2.fit.qut.edu.au/CompSci/PLAS//ComponentPascal/

Eiffel - http://dotnet.eiffel.com/

Haskell - http://haskell.cs.yale.edu/ghc/

Mercury - http://www.cs.mu.oz.au/research/mercury/

Oberon - http://www.oberon.ethz.ch/lightning/

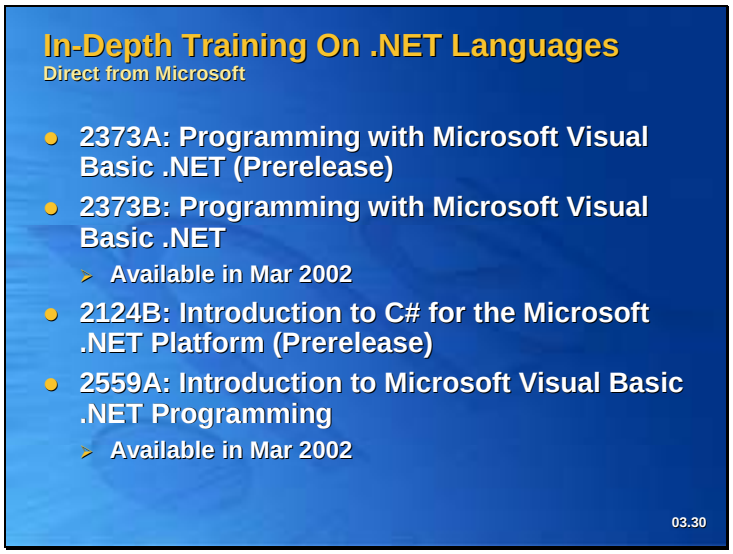
Mondrian - http://www.mondrian-script.org

Scheme - http://rover.cs.nwu.edu/~scheme/

Smalltalk - http://www.qks.com/

Standard ML - http://research.microsoft.com/Projects/SML.NET/index.htm

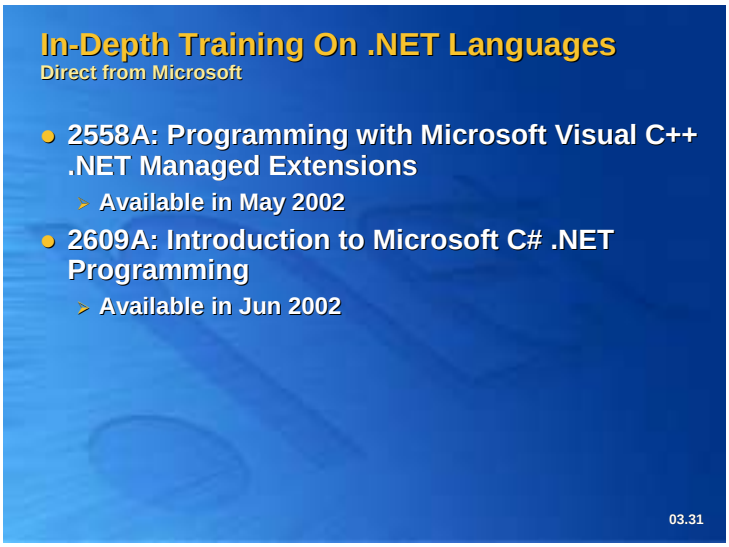
TMT Pascal - http://www.tmt.com/net.htm


In-Depth Training On .NET Languages
Direct from Microsoft
2373A: Programming with Microsoft Visual Basic .NET (Prerelease)
2373B: Programming with Microsoft Visual Basic .NET
Available in Mar 2002
2124B: Introduction to C# for the Microsoft .NET Platform (Prerelease)
2559A: Introduction to Microsoft Visual Basic .NET Programming
Available in Mar 2002
03.30

In-Depth Training On .NET Languages

Direct from Microsoft

 ©2002 Microsoft Corporation


In-Depth Training On .NET Languages
Direct from Microsoft
2558A: Programming with Microsoft Visual C++ .NET Managed Extensions
Available in May 2002
2609A: Introduction to Microsoft C# .NET Programming
Available in Jun 2002
03.31

In-Depth Training On .NET Languages

Direct from Microsoft

 ©2002 Microsoft Corporation